Heuristic Algorithms for Combinatorial Optimization problems

Ph.D. course in Computer Science

Roberto Cordone

DI - Università degli Studi di Milano

E-mail: roberto.cordone@unimi.it

Web page: https://homes.di.unimi.it/cordone/courses/2023-haco/2023-haco.html

Overcoming local optima

The *steepest descent* exchange heuristics only provide local optima

In order to improve, one can

- repeat the search (*How to avoid following the same path?*)
- extend the search (*How to avoid falling in the same optimum?*)

In the constructive algorithms only repetition was possible

The constructive metaheuristics exploit

- randomization
- memory

to operate on $\Delta_A^+(x)$ and $\varphi_A(i, x)$

The exchange metaheuristics exploit them to operate on

1. the starting solution $x^{(0)}$ (multi-start, *ILS*, *VNS*)
2. the neighbourhood $N(x)$ (*VND*)
3. the selection criterium $\varphi(x, A, D)$ (*DLS*/*GLS*)
4. the selection rule $\arg\min$ (*SA*, *TS*)

Modify the starting solution

It is possible to create different starting solutions

- generating them at random
  - with uniform probability
  - with biased distributions (*based on the data, possibly on memory*)
- applying different constructive algorithms
  - heuristics
  - metaheuristics (*with randomisation and/or memory*)
- applying the exchange algorithm to modify the solutions visited
  (*therefore with memory, and usually also randomisation*)

Modify the starting solution: random generation

The advantages of random generation are

- conceptual simplicity
- quickness for the problems in which it is easy to guarantee feasibility
- control on the probability distribution in X based on
  - element cost (e.g., favour the cheapest elements)
  - element frequency during the past search,
    to favour the most frequent elements (intensification)
    or the less frequent ones (diversification)

  This combines randomization and memory
- asymptotic convergence to the optimum (in infinite time)

The disadvantages of random generation are

- scarce quality of the starting solutions (*not the final ones!*)
- long times before reaching the local optimum

  This depends on the complexity of the exchange algorithm
- inefficiency when deciding feasibility is $\mathcal{NP}$-complete

Modify the starting solution: constructive procedures

Multi-start methods are the classical approach

- design several constructive heuristics
- each constructive heuristic generates a starting solution
- each starting solution is improved by the exchange heuristic

The disadvantages are

1. scarce control: the generated solutions tend to be similar
2. impossibility to proceed indefinitely: the number of repetitions is fixed
3. high design effort: several different algorithms must be designed
4. no guarantee of convergence, not even in infinite time

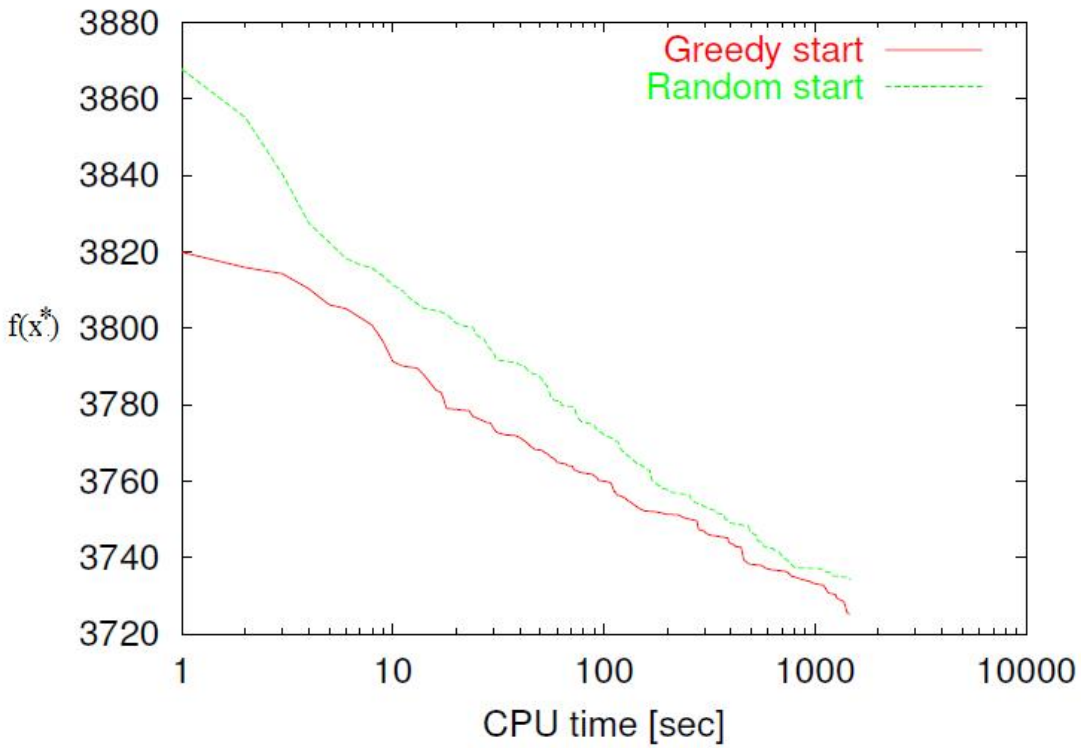
Consequently, constructive metaheuristics are preferred nowadays

GRASP and Ant System include by definition an exchange procedure

Influence of the starting solution

If the exchange heuristic is

- good, the starting solution has a short-lived influence:
  a random or heuristic generation of $x^{(0)}$ are very similar
- bad, the starting solution has a long-lived influence:
  a good heuristic to generate $x^{(0)}$ is useful



This exchange heuristic is not very good

Modify the starting solution exploiting the previous ones

The idea is to exploit the information on previously visited solutions

- save reference solutions, such as the best local optimum found so far and possibly other local optima
- generate the new starting solution modifying the reference ones

The advantages of this approach are

- control: the modification can be reduced or increased *ad libitum*
- good quality: the starting solution is very good
- conceptual simplicity
- implementation simplicity: the modification can be performed with the operations defining the neighbourhood
- asymptotic convergence to the optimum under suitable conditions

Iterated Local Search (*ILS*)

The Iterated Local Search (*ILS*), proposed by Lourenço, Martin and Stützle (2003) requires

- a *steepest descent* exchange heuristic to produce local optima
- a perturbation procedure to generate the starting solutions
- an acceptance condition to decide whether to change the reference solution x
- a termination condition

Algorithm IteratedLocalSearch$(I, x^{(0)})$

$x :=$ SteepestDescent$(x^{(0)})$; $x^* := x$;

For $l := 1$ *to* ℓ *do*

  $x' :=$ Perturbate(x);

  $x' :=$ SteepestDescent(x');

  If Accept(x', x^*) *then* $x := x'$;

  If $f(x') < f(x^*)$ *then* $x^* := x'$;

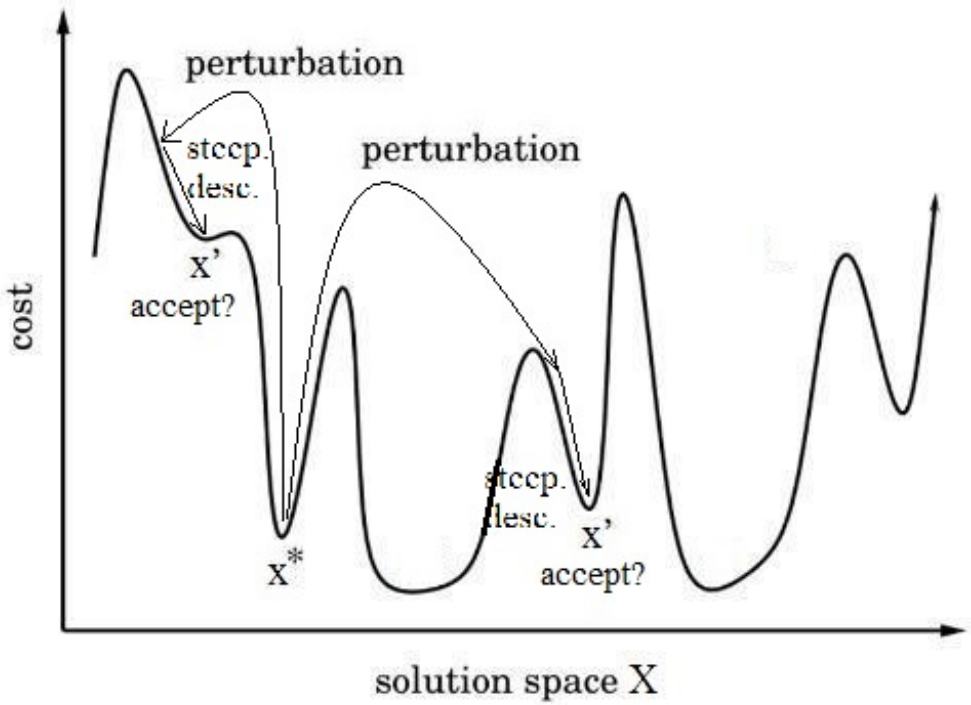
EndFor;

Return $(x^*, f(x^*))$;

Iterated Local Search (*ILS*)

The idea is that

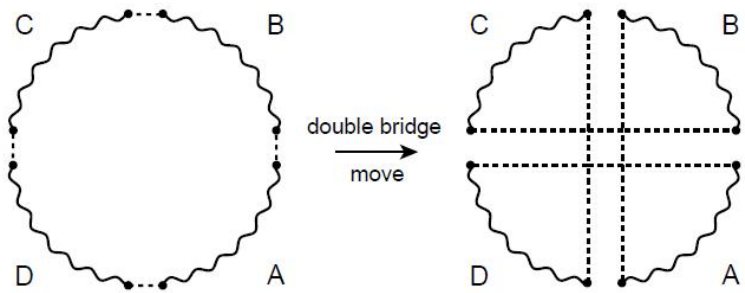
- the exchange heuristic quickly explores an attraction basin, terminating into a local optimum
- the perturbation procedure moves to another attraction basin
- the acceptance condition evaluates if the new local optimum is a promising starting point for the following perturbation



Example: *ILS* for the *TSP*

A classical application of *ILS* to the *TSP* uses

- exchange heuristic: *steepest descent* with neighbourhood $N_{\mathcal{R}_2}$
- perturbation procedure: a *double-bridge* move
  that is particular kind of 4-exchange



- acceptance condition: the best known solution improves

$$f(x') < f(x^*)$$

Perturbation procedure

Let $\mathcal{O}$ be the operation set that defines neighbourhood $N_{\mathcal{O}}$

The perturbation procedure performs a random operation o

- with $o \in \mathcal{O}' \not\subseteq \mathcal{O}$, to avoid that the exchange heuristic drive solution x' back to the starting local optimum x

Two typical definitions of $\mathcal{O}'$ are

- sequences of $k > 1$ operations of $\mathcal{O}$
  (generating a random sequence is cheap)
- conceptually different operations
  (e.g., vertex exchanges instead of edge exchanges)

The main difficulty of *ILS* is in tuning the perturbation: if it is

- too strong, it turns the search into a random restart
- too weak, it guides the search back to the starting local optimum
  - wasting time
  - possibly losing the asymptotic convergence

Ideally one would like to enter any basin and get out of any basin

Acceptance condition

Algorithm IteratedLocalSearch$(I, x^{(0)})$

$x :=$ SteepestDescent$(x^{(0)})$; $x^* := x$;

For $I := 1$ *to* ℓ *do*

  $x' :=$ Perturbate(x);

  $x' :=$ SteepestDescent(x');

  If Accept(x', x^*) *then* $x := x'$;

  If $f(x') < f(x^*)$ *then* $x^* := x'$;

EndWhile;

Return $(x^*, f(x^*))$;

The acceptance condition balances intensification and diversification

- accepting only improving solutions favours intensification

  $$\text{Accept}(x', x^*) := (f(x') < f(x^*))$$

  The reference solution is always the best found: $x = x^*$

- accepting any solution favours diversification

  $$\text{Accept}(x', x^*) := \textit{true}$$

  The reference solution is always the last optimum found: $x = x'$

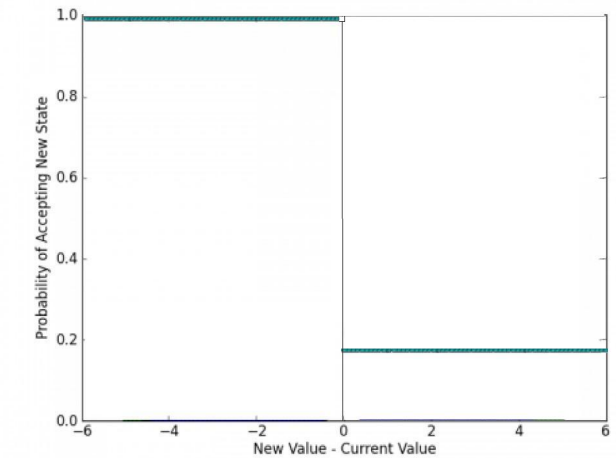
Acceptance condition

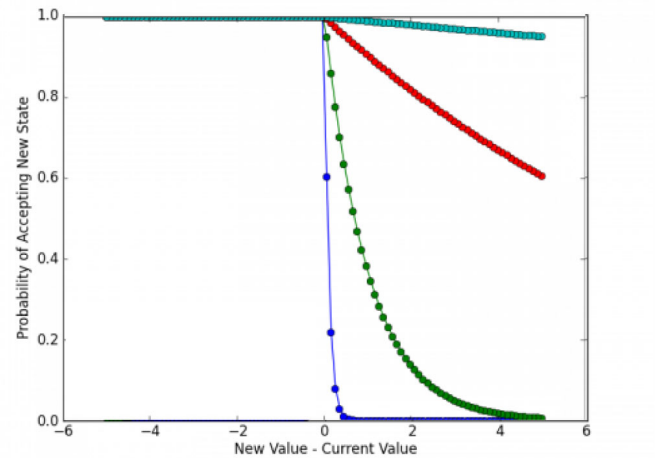
Intermediate strategies can be defined based on $\delta f = f(x') - f(x^*)$

- if $\delta f < 0$, always accept x'
- if $\delta f \geq 0$, accept x' with probability $\pi(\delta f)$, where $\pi(\cdot)$ is a nonincreasing function

The most typical cases are:

- constant probability: $\pi(\delta f) = \bar{\pi} \in (0; 1)$ for each $\delta f \geq 0$
- monotonically decreasing probability with $\pi(0) = 1$ and $\lim_{\delta f \to +\infty} \pi(\delta) = 0$





Memory can also be used, accepting x' more easily if many iterations have elapsed since the last improvement of x^*

Variable Neighbourhood Search (*VNS*)

A method very similar to *ILS* is the *Variable Neighbourhood Search* proposed by Hansen and Mladenović (1997)

The main differences betwween *ILS* and *VNS* are the use of

- the strict acceptance condition: $f(x') < f(x^*)$
- an adaptive perturbation mechanism instead of the fixed one

VNS often introduces also neighbourhood modifications (*later on this*)

The perturbation mechanism is based on a hierarchy of neighbourhoods, that is a family of neighbourhoods with an increasing parametric size s

$$N_1 \subset N_2 \subset \ldots \subset N_s \subset \ldots N_{s_{max}}$$

Typically one uses the parameterised neighbourhoods

- N_{H_s}, based on the Hamming distance between subsets
- $N_{\mathcal{O}_s}$, based on the sequences of operations from a basic set $\mathcal{O}$

and extracts $x^{(0)}$ randomly from a neighbourhood of the hierarchy

Adaptive perturbation mechanism

It is called *variable neighbourhood* because the neighbourhood used to extract $x^{(0)}$ varies based on the results of the exchange heuristic

- if a better solution is found, use the smallest neighbourhood, to generate a starting solution very close to x^* (intensification)
- if a worse solution is found, use a slightly larger neighbourhood, to generate a starting solution slightly farther from x^* (diversification)

The method has three parameters

1. $s_{\min}$ identifies the smallest neighbourhood to generate new solutions
2. $s_{\max}$ identifies the largest neighbourhood to generate new solutions
3. δs is the increase of s between two subsequent attempts

The exchange heuristic adopts the smallest neighbourhood to be efficient

(N_1, *or anyway* N_s *with* $s \leq s_{\min}$)

General scheme of the *VNS*

Algorithm VariableNeighbourhoodSearch(I, $x^{(0)}$, $s_{\min}$, $s_{\max}$, δs)

$x :=$ SteepestDescent($x^{(0)}$); $x^* := x$;

$s := s_{\min}$;

For $l := 1$ *to* ℓ *do*

  $x' :=$ Shaking(x^*, s);

  $x' :=$ SteepestDescent(x');

  If $f(x') < f(x^*)$

    then $x^* := x'$; $s := s_{\min}$;

    else $s := s + \delta s$;

  If $s > s_{\max}$ *then* $s := s_{min}$;

EndWhile;

Return $(x^*, f(x^*))$;

- the reference solution x' is always the best known solution x^*
- the starting solution is obtained extracting it at random from the current neighbourhood of the reference solution $N_s(x^*)$
- the exchange heuristic produces a local optimum with respect to the basic neighbourhood N
- if the best known solution improves, the current neighbourhood becomes $N_{s_{\min}}$
- otherwise, move to a larger neighbourhood $N_{s+\delta s}$, never exceeding $N_{s_{\max}}$

Tuning of the shaking parameters

The value of $s_{\min}$ must be

- large enough to get out of the current attraction basin
- small enough to avoid jumping over the adjacent attraction basins

In general, one sets $s_{\min} = 1$, and increases it if experimentally profitable

The value of $s_{\max}$ must be

- large enough to reach any useful attraction basin
- small enough to avoid reaching useless regions of the solution space

Example: the diameter of the search space for the basic neighbourhood: $\min(k, n-k)$ for the *MDP*; n for the *TSP* and *MAX-SAT*, etc...

The value of δs must be

- large enough to reach $s_{\max}$ in a reasonable time
- small enough to allow each reasonable value of s

In general, one sets $\delta s = 1$, unless $s_{\max} - s_{\min}$ is too large

Skewed VNS

In order to favour diversification, it is possible to accept x' when

$$f(x') < f(x^*) + \alpha \, d_H(x', x^*)$$

where

- $d_H(x', x^*)$ is the Hamming distance fra x' and x^*
- $\alpha > 0$ is a suitable parameter

This allows to accept worsening solutions as long as they are far away

- $\alpha \approx 0$ tends to accept only improving solutions
- $\alpha \gg 0$ tends to accept any solution

Of course, the random strategies seen for the ILS can also be adopted

Extending the local search without worsening

Instead of repeating the local search, extend it beyond the local optimum

To avoid worsening solutions, the selection step must be modified

$$\tilde{x} := \arg \min_{x' \in N(x)} f(x')$$

and two main strategies allow to do that

- the *Variable Neighbourhood Descent* (*VND*) changes the neighbourhood N
  - it guarantees an evolution with no cycles (*the objective improves*)
  - it terminates when all neighbourhoods have been exploited
- the *Dynamic Local Search* (*DLS*) changes the objective function f ($\tilde{x}$ *is better than* x *for the new objective, possibly worse for the old*)
  - it can be trapped in loops (*the new objective changes over time*)
  - it can proceed indefinitely

Variable Neighbourhood Descent (VND)

The *Variable Neighbourhood Descent* of Hansen and Mladenović (1997) exploits the fact that a solution is locally optimal for a specific neighbourhood

- a local optimum can be improved using a different neighbourhood

Given a family of neighbourhoods $N_1, \ldots, N_{s_{\text{tot}}}$

1. set $s := 1$
2. apply a *steepest descent* exchange heuristic and find a local optimum $\bar{x}$ with respect to N_s
3. flag all neighbourhoods for which $\bar{x}$ is locally optimal and update s
4. if $\bar{x}$ is a local optimum for all N_s, terminate; otherwise, go back to point 2

Algorithm VariableNeighbourhoodDescent($I, x^{(0)}$)
$\text{flag}_s :=$ false $\forall k$;
$\bar{x} := x^{(0)}$; $x^* := x^{(0)}$; $s := 1$;
While $\exists s : \text{flag}_s =$ false *do*
  $\bar{x} :=$ SteepestDescent($\bar{x}, s$); { possibly truncated }
  $\text{flag}_s :=$ true;
  If $f(\bar{x}) < f(x^*)$
    then $x^* := \bar{x}$; $\text{flag}_{s'} :=$ false $\forall s' \neq s$;
  $s :=$ *Update*(s);
EndWhile;
Return $(x^*, f(x^*))$;

Anticipated termination of Steepest Descent

Using many neighbourhoods means that some might be

- rather large
- slow to explore

In order to increase the efficiency of the method one can

- adopt a first-best strategy in the larger neighbourhoods
- terminate the Steepest Descent before reaching a local optimum (*possibly even after a single step*)

Larger neighbourhoods aim to move out of the basins of attraction of smaller neighbourhoods

VND and *VNS*

There is of course a strict relation between *VND* and *VNS*

(*in fact, they were proposed in the same paper*)

The fundamental differences are that in the *VND*

- at each step the current solution is the best known one
- the neighbourhoods are explored,
  instead of being used to extract random solutions

  They are never huge
- the neighbourhoods do not necessarily form a hierarchy

  The update of s is not always an increment
- when a local optimum for each N_s has been reached, terminate

  VND is deterministic and would not find anything else

Neighbourhood update strategies for the *VND*

There are two main classes of *VND* methods

- methods with heterogeneous neighbourhoods
  - exploit the potential of topologically different neighbourhoods (e.g., exchange vertices instead of edges)

  Consequently, s periodically scans the values from 1 to s_{tot} (*possibly randomly permuting the sequence at each repetition*)
- methods with hierarchical neighbourhoods ($N_1 \subset \ldots \subset N_{s_{\text{tot}}}$)
  - fully exploit the small and fast neighbourhoods
  - resort to the large and slow ones only to get out of local optima (*usually terminating SteepestDescent prematurely*)

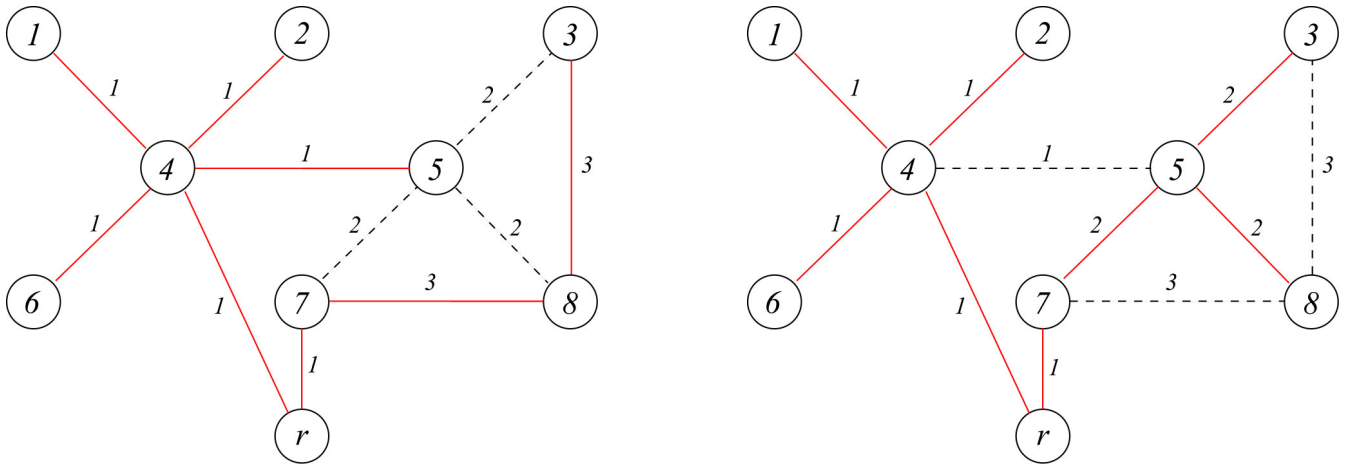
  Consequently, the update of s works as in the *VNS*
  - when no improvements can be found in N_s, increase s
  - when improvements can be found in N_s, decrease s back to 1

Terminate when the current solution is a local optimum for all N_s

- in the heterogeneous case, terminate when all fail
- in the hierarchical case, terminate when the largest fails

Example: the CMSTP

This instance of *CMSTP* has $n = 9$ vertices, uniform weights ($w_v = 1$), capacity $W = 5$ and the reported costs (the missing edges have $c_e \gg 3$)



Consider neighbourhood N_{S_1} (single-edge swaps) for the first solution:

- no edge in the right branch can be deleted
  because the left branch has zero residual capacity
  and a direct connection to the root would increase the cost
- deleting any edge in the left branch increases the total cost

The solution is a local optimum for N_{S_1}

Neighbourhood N_{T_1} (single-vertex transfers) has an improving solution, obtained moving vertex 5 from the left branch to the right one

Dynamic Local Search (DLS)

The *Dynamic Local Search* is also known as *Guided Local Search*

Its approach is complementary to *VND*

- it keeps the starting neighbourhood
- it modifies the objective function

It is often used when the objective is useless because it has wide *plateaus*

The basic idea is to

- define a penalty function $w : X \to \mathbb{N}$
- build an auxiliary function $\tilde{f}(f(x), w(x))$
  which combines the objective function f with the penalty w
- apply a *steepest descent* exchange heuristic to optimise $\tilde{f}$
- at each iteration update the penalty w based on the results

The penalty is adaptive in order to move away from recent local optima but this introduces the risk of cycling

General scheme of the *DLS*

Algorithm DynamicLocalSearch($I, x^{(0)}$)
$w :=$ StartingPenalty(I);
$\bar{x} := x^{(0)}$; $x^* := x^{(0)}$;
While Stop() $=$ *false do*
  $(\bar{x}, x_f) :=$ SteepestDescent($\bar{x}, f, w$); { possibly truncated }
  If $f(x_f) < f(x^*)$ *then* $x^* := x_f$;
  $w :=$ *UpdatePenalty*($w, \bar{x}, x^*$);
EndWhile;
Return $(x^*, f(x^*))$;

Notice that the *steepest descent* heuristic

- optimises a combination $\tilde{f}$ of f and w
- returns two solutions:
  1. a final solution $\bar{x}$, locally optimal with respect to $\tilde{f}$, to update w
  2. a solution x_f, that is the best it has found with respect to f

Variants

The penalty can be applied (for example)

- additively to the elements of the solution:

$$\tilde{f}(x) = f(x) + \sum_{i \in x} w_i$$

- multiplicatively to components of the objective $f(x) = \sum_j \phi_j(x)$:

$$\tilde{f}(x) = \sum_j w_j \, \phi_j(x)$$

The penalty can be updated

- at each single neighbourhood exploration
- when a local optimum for $\tilde{f}$ is reached
- when the best known solution x^* is unchanged for a long time

The penalty can be modified with

- random updates: "noisy" perturbation of the costs
- memory-based updates, favouring the most frequent elements (intensification) or the less frequent ones (diversification)

Example: *DLS* for the *MCP*

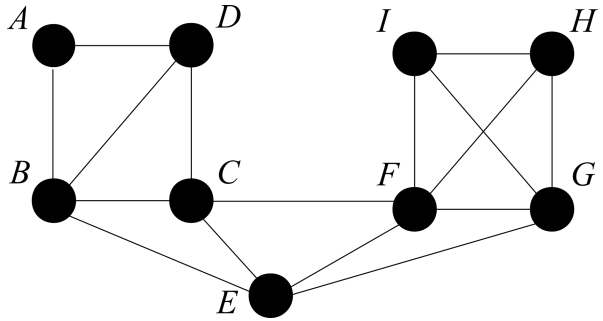
Given a undirected graph, find a maximum cardinality clique

- the exchange heuristic is a *VND* using the neighbourhoods
  1. N_{A_1} (vertex addition): the solution always improves, but the neighbourhood is very small and often empty
  2. N_{S_1} (exchange of an internal vertex with an external one): the neighbourhood is larger, but forms a *plateau* (uniform objective)
- the objective provides no useful direction in either neighbourhood
- associate to each vertex i a penalty w_i initially equal to zero
- the exchange heuristic minimises the total penalty (within the neighbourhood!)
- update the penalty
  1. when the exploration of N_{S_1} terminates: the penalty of the current clique vertices increases by 1
  2. after a given number of explorations: all the nonzero penalties decrease by 1

The rationale of the method consists in aiming to

- expel the internal vertices (diversification)
- in particular, the oldest internal vertices (memory)

Example: *DLS* for the *MCP*



Start from $x^{(0)} = \{B, C, D\}$, with $w = [\,0\,1\,1\,1\,0\,0\,0\,0\,0\,]$

1. $w(\{B, C, E\}) = w(\{A, B, D\}) = 2$, but $\{A, B, D\}$ wins lexicographically: $x^{(1)} = \{A, B, D\}$ with $w = [\,1\,2\,1\,2\,0\,0\,0\,0\,0\,]$
2. $x^{(2)} = \{B, C, D\}$ with $w = [\,1\,3\,2\,3\,0\,0\,0\,0\,0\,]$ is the only neighbour
3. $w(\{B, C, E\}) = 5 < 7 = w(\{A, B, D\})$: $x^{(3)} = \{B, C, E\}$ with $w = [\,1\,4\,3\,3\,1\,0\,0\,0\,0\,]$
4. $w(\{C, E, F\}) = 4 < 10 = w(\{B, C, D\})$: $x^{(4)} = \{C, E, F\}$ with $w = [\,1\,4\,4\,3\,2\,1\,0\,0\,0\,]$
5. $w(\{E, F, G\}) = 3 < 11 = w(\{B, C, E\})$: $x^{(5)} = \{E, F, G\}$ with $w = [\,1\,4\,4\,3\,3\,2\,1\,0\,0\,]$
6. $w(\{F, G, H\}) = w(\{F, G, I\}) = 3 < 9 = w(\{C, E, F\})$: $x^{(6)} = \{F, G, H\}$ with $w = [\,1\,4\,4\,3\,3\,3\,2\,1\,0\,]$

Now the neighbourhood N_{A_1} is not empty: $x^{(7)} = \{F, G, H, I\}$

Example: *DLS* for the *MAX-SAT*

Given m logical disjunctions depending on n logical variables, find a truth assignment satisfying the maximum number of formulae

- neighbourhood N_{F_1} (1-flip) is generated complementing a variable
- associate to each logical formula a penalty w_j initially equal to 1 (*each component is a satisfied formula*)
- the exchange heuristic maximizes the weight of satisfied formulae thus modifying their number with the multiplicative penalty
- the penalty is updated
  1. increasing the weight of unsatisfied formulae to favour them

     $$w_j := \alpha_{\text{us}}\ w_j \text{ for each } j \in U(x) \quad (\text{with } \alpha_{\text{us}} > 1)$$

     when a local optimum is reached
  2. reducing the penalty towards 1

     $$w_j := (1-\rho)\ w_j + \rho \cdot 1 \text{ for each } j \in C \quad (\text{with } \rho \in (0,1))$$

     with a certain probability or after a certain number of updates

Example: *DLS* for the *MAX-SAT*

The rationale of the method consists in aiming to

- satisfy the currently unsatisfied formulae (diversification)
- in particular, those which have been unsatisfied for longer time and more recently (memory)

The parameters tune intensification and diversification

- small values of α_{us} and ρ preserve the current penalty (intensification)
- large values of α_{us} and ρ cancel the current penalty (diversification)

Extending the local search with worsenings

If the neighbourhood and objective remain the same,
the rule of acceptance must change: instead of

$$x' := \arg \min_{x \in N(x)} f(x)$$

select a nonminimal (possibly, even nonimproving) solution

The main problem is the risk of cyclically visiting the same solutions

The two main strategies that allow to control this risk are

- *Simulated Annealing* (*SA*), which uses randomization to make repetitions unlikely
- *Tabu Search* (*TS*), which uses memory to forbid repetitions

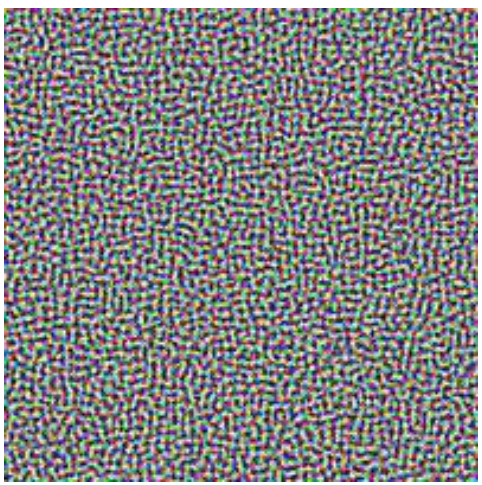
Annealing

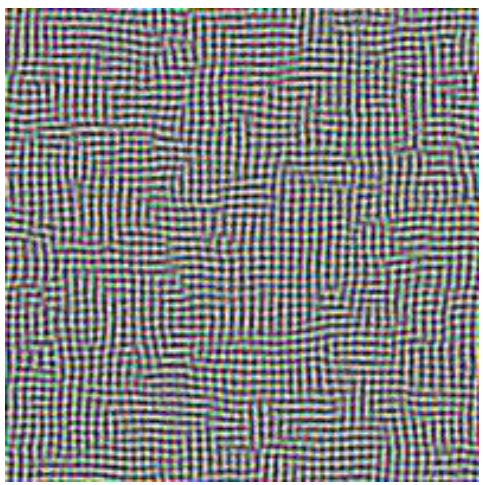
The *SA* derives from Metropolis' algorithm (1953), which aims to simulate the "annealing" process of metals:

- bring the metal to a temperature close to fusion, so that its particles distribute at random
- cool the metal very slowly, so that the energy decreases, but in a time sufficiently long to converge to thermal equilibrium

The aim of the process is to obtain

- a very regular and defectless crystal lattice, that corresponds to the base state (minimum energy configuration)
- a material with useful physical properties





Simulation and optimization

The situation has similarities with Combinatorial Optimization problems

- the states of the physical system correspond to the solutions
- the energy corresponds to the objective function
- the base state corresponds to the globally optimal solutions (minima)
- the state transitions correspond to local search moves
- the temperature corresponds to a numerical parameter

This suggests to use Metropolis' algorithm for optimization

According to thermodynamics at the thermal equilibrium
the probability of observing each state i depends on its energy E_i

$$\pi'_T(i) = \frac{e^{\frac{-E_i}{k\,T}}}{\sum_{j \in S} e^{\frac{-E_j}{k\,T}}}$$

where S is the state set, T the temperature and k Boltzmann's constant

It is a dynamic equilibrium, with ongoing state transitions in all directions

Metropolis' algorithm

Metropolis' algorithm generates a random sequence of states

- the current state i has energy E_i
- the algorithm perturbs i, generating a state j with energy E_j
- the current state moves from i to j with probability

$$\pi_T(i,j) = \begin{cases} 1 & \text{if } E_j < E_i \\ e^{\frac{E_i - E_j}{k\,T}} = \frac{\pi'(j)}{\pi'(i)} & \text{if } E_j \geq E_i \end{cases}$$

that is the transition is

- deterministic if improving (because that is the final purpose)
- based on the conditional probability if worsening

Simulated Annealing applies exactly the same principle

General scheme of *Simulated Annealing*

Algorithm SimulatedAnnealing$(I, x^{(0)}, T^{[0]})$

$x := x^{(0)}$; $x^* := x^{(0)}$; $T := T^{[0]}$;

While Stop() $=$ *false do*

$x' :=$ RandomExtract(N, x); { random uniform extraction }

If $f(x') < f(x)$ *or* $U[0;1] \leq e^{\frac{f(x)-f(x')}{T}}$ *then* $x := x'$;

If $f(x') < f(x^*)$ *then* $x^* := x'$;

$T :=$ Update(T);

EndWhile;

Return $(x^*, f(x^*))$;

As the neighbourhood is used to generate a solution (not fully explored), it is possible to worsen even if improving solutions exist

A precomputed table of values for $e^{\frac{\delta f}{T}}$ can improve the efficiency

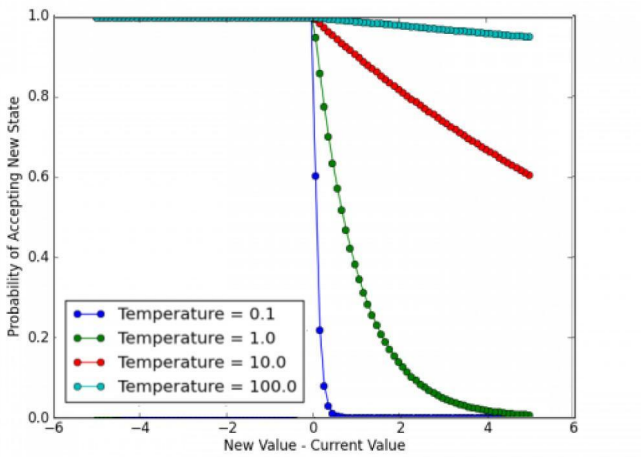
Several update schemes can be designed for the "temperature" T

Acceptance criterium

T rules the probability to accept worsenings

$$\pi_T(x, x') = \begin{cases} 1 & \text{if } f(x') < f(x) \\ e^{\frac{f(x)-f(x')}{T}} & \text{if } f(x') \geq f(x) \end{cases}$$

- $T \gg 0$ diversifies because nearly all solutions are accepted: in the extreme case, it is a *random walk*
- $T \approx 0$ intensifies nearly all worsening solutions are rejected: in the extreme case, it is a *steepest descent*



Notice the similarity with the ILS

Asymptotic convergence to the optimum

Due to the acceptance rule, the current solution x is a random variable: its "state probability" $\pi'(x)$ combines on all possible predecessors $x^{(t-1)}$

- the "state probability" $\pi'\left(x^{(t-1)}\right)$ of the predecessor
- the probability to choose the move from $x^{(t-1)}$ to x, that is uniform
- the probability to accept the move, that is

$$\pi_T\left(x^{(t-1)}, x\right) = \begin{cases} 1 & \text{if } f(x) < f\left(x^{(t-1)}\right) \\ e^{\frac{f(x^{(t-1)}) - f(x)}{T}} & \text{if } f(x) \geq f\left(x^{(t-1)}\right) \end{cases}$$

As it depends only on the previous step, the solution is a Markov chain

For fixed temperature T, the transition probabilities are stationary: it is a homogeneous Markov chain

If the search space for neighbourhood N is connected, the probability to reach each state is > 0: it is an irreducible Markov chain

Under these assumptions, the state probability converges to a stationary distribution independent from the starting state

Asymptotic convergence to the optimum

The stationary distribution favours "good" solutions with the same law imposed by thermodynamics on physical systems at thermal equilibrium

$$\pi_T(x) = \frac{e^{\frac{-f(x)}{T}}}{\sum_{x \in X} e^{\frac{-f(x)}{T}}} \quad \text{for each } x \in X$$

where X is the feasible region and T the "temperature" parameter

The distribution converges to a limit distribution as $T \to 0$

$$\pi(x) = \lim_{T \to 0} \pi_T(x) = \begin{cases} \frac{1}{|X^*|} & \text{for } x \in X^* \\ 0 & \text{for } x \in X \setminus X^* \end{cases}$$

which corresponds to a certain convergence to a globally optimal solution

Asymptotic convergence to the optimum

This result however holds at the equilibrium, in infinite time

In practice, low values of T imply

- a high probability to visit a global optimum, but also
- a slow convergence to the optimum (*many exchanges are rejected*)

In a finite time, the result obtained with low T can be far from optimal

Hence, T starts high and is progressively updated decreasing over time

The starting value $T^{[0]}$ should be

- high enough to allow to reach any solution quickly
- small enough to discourage visiting very bad solutions

A classical tuning for $T^{[0]}$ is to

- sample the first neighbourhood $N\left(x^{(0)}\right)$
- set $T^{[0]}$ such as to accept a fraction α of the sampled solutions

Temperature update

The temperature is updated by subsequent phases ($r = 0, \ldots, m$)

- each phase applies a constant value $T^{[r]}$ for $\ell^{[r]}$ iterations
- $T^{[r]}$ decreases exponentially from phase to phase

$$T^{[r]} := \alpha T^{[r-1]} = \alpha^r T^{[0]}$$

- $\ell^{[r]}$ increases from phase to phase (often linearly)
  with values related to the diameter of the search graph
  (therefore to the size of the instance)

Since T is variable, the Markov chain x is not homogeneous, but

- if T decreases slowly enough, it converges to the global optimum
- good parameters to tune the decrease depend on the instance
  (namely, on $f(\tilde{x}) - f(x^*)$, where $f(\tilde{x})$ is the second best value of f)

But the best parameter values are not known a priori

Adaptive SA variants tune the temperature T based on the results

- set T to a value such that a given fraction of $N(x)$ is accepted
- increase T if the solution has not improved for a certain time
  (diversification); otherwise decrease it (intensification)

Tabu Search

The *Tabu Search* (*TS*) has been proposed by Glover (1986)

It keeps the basic selection rule of *steepest descent*

$$x' := \arg \min_{x \in N(x)} f(x)$$

without the termination condition

But this implies cycling!

The *TS* imposes a tabu to forbid the solutions already visited

$$x' := \arg \min_{x \in N(x) \setminus X_V} f(x)$$

where X_V is the set of the already visited solutions

A simple idea, but how to manage the tabu efficiently and effectively?

Exchange heuristics with tabu

An exchange heuristic that explores a neighbourhood imposing a tabu on the already visited solutions requires to:

1. evaluate the feasibility of each subset produced by the exchanges (unless guaranteed *a priori*)
2. evaluate the cost of each feasible solution
3. evaluate the tabu status of each feasible promising solution

in order to select the feasible best nontabu solution

An elementary way to implement the evaluation of the tabu is

- save the visited solutions in a suitable structure (tabu list)
- check each explored solution making a query on the tabu list

Potential inefficiency of the tabu mechanism

This elementary evaluation of the tabu however is very inefficient

- the comparison of the solutions at step t requires time $O(t)$
  (*reducible with hash tables or search trees*)
- the number of solutions visited grows indefinitely over time
- the memory occupation grows indefinitely over time

The *Cancellation Sequence Method* and the *Reverse Elimination Method* tackle these problems, exploiting the fact that in general

- the solutions visited form a chain with small variations
- few solutions visited are neighbours of the current one

The idea is to focus on variations

- save move lists, instead of solutions
- evaluate the overall performed variations, instead of the single moves
- find the solutions which have undergone small overall variations
  (recent ones or submitted to variations subsequently reversed)

Potential ineffectiveness of the tabu mechanism

Other subtle phenomena influence the effectiveness of the method

Forbidding the solutions visited can have two different negative effects:

- it can disconnect the search graph,
  creating impassable "iron curtains" that block the search
  (*the prohibition should not be permanent*)
- it can slow down the exit from attraction basins,
  creating a "gradual filling" effect that slows down the search
  (*the prohibition should be extended*)

The two phenomena suggest apparently opposite remedies

How to combine them?

Example

A very degenerate example is provided by the following problem

- the ground set $B = \{1, \ldots, n\}$ includes the first n natural numbers
- all subsets are feasible: $X = 2^B$
- the objective combines a nearly uniform additive term $\phi_i = 1 + \epsilon i$ $(0 < \epsilon \ll 1)$ and (only if $x = x^*$) a strong negative term

$$f(x) = \begin{cases} \sum_{i \in x} (1 + \epsilon i) & \text{for } x \neq x^* \\ \sum_{i \in x} (1 + \epsilon i) - n - 1 & \text{for } x = x^* \end{cases}$$

where x^* is suitably chosen in X

Using the neighbourhood of all solutions at Hamming distance ≤ 1

$$N_{H_1}(x) = \left\{ x' \in 2^B : d_H(x, x') \leq 1 \right\}$$

the problem has

- a global optimum x^*, with $f(x^*) = n(n+1)\epsilon/2 - 1 < 0$, whose attraction basin includes the solutions x with $d_H(x, x^*) \leq 1$
- a local optimum $\bar{x} = \emptyset$ with $f(\bar{x}) = 0$, whose attraction basin includes the solutions x with $d_H(x, x^*) > 1$

Example

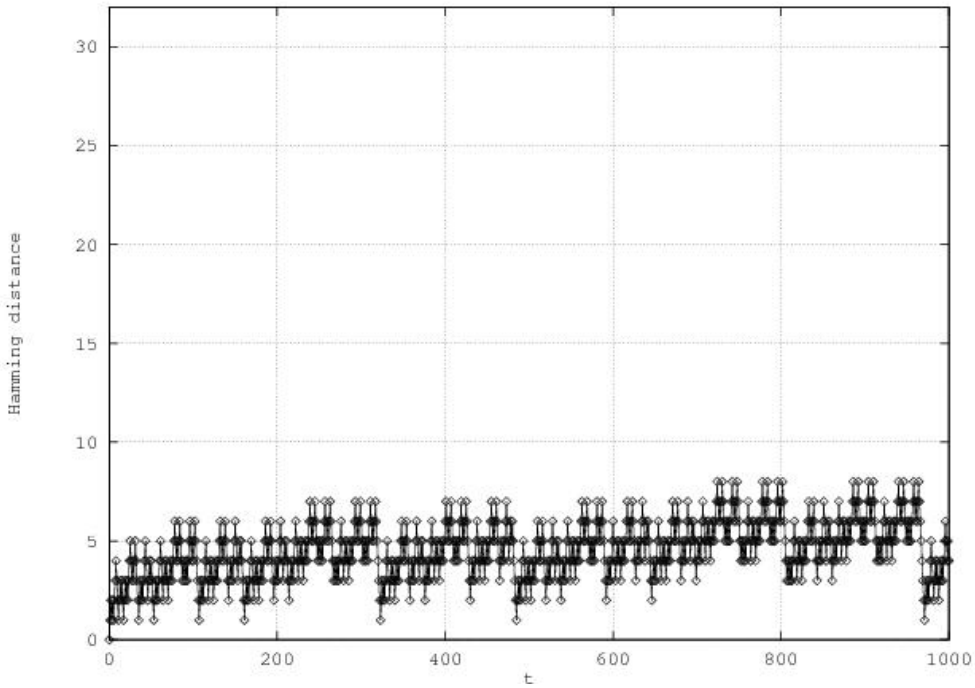
Starting from $x^{(0)} = \bar{x} = \emptyset$ and forbidding all the solutions visited:

- visit methodically most of 2^B, with f and $d(x, \bar{x})$ going up and down
- for $4 \leq n \leq 14$ the search graph is disconnected and the search is stuck (1011 can't be reached), but all solutions are at least explored
- for $n \geq 15$, the search is stuck and some unvisited solutions are not explored, possibly missing the optimum

t	f	x	$d(x, \bar{x})$
1	0	0000	0
2	$1+\epsilon$	1000	1
3	$2+3\epsilon$	1100	2
4	$1+2\epsilon$	0100	1
5	$2+5\epsilon$	0110	2
6	$1+3\epsilon$	0010	1
7	$2+4\epsilon$	1010	2
8	$3+6\epsilon$	1110	3
9	$4+10\epsilon$	1111	4
10	$3+9\epsilon$	0111	3
11	$2+7\epsilon$	0011	2
12	$1+4\epsilon$	0001	1
13	$2+5\epsilon$	1001	2
14	$3+7\epsilon$	1101	3
15	$2+6\epsilon$	0101	2

Example

The objective function profile confirms the limitations of the method



The solution x repeatedly gets far from $x^{(0)} = \bar{x}$ and close to it

- it visits nearly the whole attraction basin of $\bar{x}$
- in the end, it does not get out of it, but gets stuck in a solution whose neighbourhood is fully tabu
- if it removes the oldest tabu, the exploration goes around and the risk of looping gets back

Attribute-based tabu

Some simple devices can be adopted in order to control these problems

1. forbid all solutions that share "attributes" with the visited ones, instead of forbidding only the visited solutions
   - define a set A of attributes
   - define for each solution $x \in X$ a subset of attributes $A_x \subseteq A$
   - declare a subset of tabu attributes $\bar{A} \subseteq A$ (empty at first)
   - forbid all the solutions with tabu attributes

$$x \text{ is tabu } \Leftrightarrow A_x \cap \bar{A} \neq \emptyset$$

   - move from the current solution x to x' such that $A_{x'} \cap \bar{A} = \emptyset$ and add to $\bar{A}$ the attributes possessed by x and not by x'

$$\bar{A} := \bar{A} \cup (A_x \setminus A_{x'})$$

(*in this way, x becomes tabu*)

This allows to

- avoid also solutions similar to the visited ones
- get more quickly far away from visited local optima

Temporary tabu and aspiration criterium

Since the tabu creates regions hard or impossible to reach

2. give a limited length L (tabu tenure) to the prohibition
   - the tabu solutions become feasible again after a while
   - the same solutions can be revisited

   (*but, if $\bar{A}$ is different, the future evolution will be different*)

Tuning the tabu tenure is fundamental for the effectiveness of *TS*

The tabu could forbid a global optimum similar to a known solution

3. introduce an aspiration criterium: a tabu solution better than the best known one is anyway accepted

   (*of course, there is no risk of looping*)

There are looser aspiration criteria, but they are not commonly used

The tabu could forbid all neighbour solutions

4. if all neighbour solutions are tabu, accept the one with the oldest tabu (it can be interpreted as another aspiration criterium)

General scheme of the *TS*

Algorithm TabuSearch$(I, x^{(0)}, L)$

$x := x^{(0)}$; $x^* := x^{(0)}$;

$\bar{A} := \emptyset$;

While Stop() $=$ *false do*

  $f' := +\infty$;

  For each $y \in N(x)$ *do*

    If $f(y) < f'$ *then*

      If Tabu$(y, \bar{A}) =$ *false or* $f(y) < f(x^*)$ *then* $x' := y$; $f' := f(y)$;

    EndIf

  EndFor

  $\bar{A} :=$ Update$(\bar{A}, x', L)$;

  If $f(x') < f(x^*)$ *then* $x^* := x'$;

EndWhile

Return $(x^*, f(x^*))$;

Tabu attributes

The concept of "attribute" is intentionally generic; the simpler ones are

- inclusion of an element in the solution ($A_x = x$):
  when the move from x to x' expels an element i from the solution, the tabu forbids the reinsertion of i in the solution
  - x has the attribute "presence of i" and x' hasn't got it
  - the attribute "presence of i" enters $\bar{A}$
  - every solution including i becomes tabu
- exclusion of an element from the solution ($A_x = B \setminus x$):
  when the move from x to x' inserts an element i into the solution, the tabu forbids the removal of i from the solution
  - x has the attribute "absence of i" and x' hasn't got it
  - the attribute "absence of i" enters $\bar{A}$
  - every solution devoid of i becomes tabu

Different attribute sets can be combined, each with its tenure and list (e.g., after replacing i with j, forbid to remove j for L^{in} steps and to insert i for L^{out} steps, with $L^{\text{in}} \neq L^{\text{out}}$)

Tabu attributes

Other (less frequent) examples of attributes

- the value of the objective function: forbid solutions of a given value, previously assumed by the objective
- the value of an auxiliary function

Complex attributes can be obtained combining simple attributes

- the coexistence in the solution of two elements (or their separation)
- or, if a move replaces element i with element j,
  the tabu can forbid the removal of j to include i,
  but allow the simple removal of j and the simple inclusion of i

Efficient evaluation of the tabu status

The evaluation of the tabu status must be efficient and avoid scanning the whole solution (as for feasibility and cost)

- the attributes are associated to moves, not to solutions: do not check whether the solution includes i, but whether the move adds i

Let T_i be the iteration when attribute $i \in A$ became tabu ($-\infty$ if $i \notin \bar{A}$)

To evaluate the tabu status in constant time simply check

$$t \leq T_i + L$$

If the tabu is on insertions ($A = x$), at iteration t

- forbid the moves that add $i \in B \setminus x$ when $t \leq T_i^{\text{in}} + L^{\text{in}}$
- update $T_i^{\text{in}} := t$ for each i removed ($i \in x \setminus x'$)

If the tabu is on deletions ($A = B \setminus x$), at iteration t

- forbid the moves that delete $i \in x$ when $t \leq T_i^{\text{out}} + L^{\text{out}}$
- update $T_i^{\text{out}} := t$ for each i added ($i \in x' \setminus x$)

As either $i \in x$ or $i \in B \setminus x$, a single vector T is enough for both checks

More sophisticated attributes require more complex structures

Example: the *TSP*

Consider the neighbourhood $N_{\mathcal{R}_2}$ generated by 2-opt exchanges and use as attributes both the presence and the absence of arcs in the solution

- at first set $T_{ij} = -\infty$ for each arc $(i,j) \in A$
- at each step t, explore the $n(n-1)/2$ pairs of removable arcs and the corresponding pairs of arcs which would replace them
- the move (i,j), which replaces (s_i, s_{i+1}) and (s_j, s_{j+1}) with (s_i, s_j) and (s_{i+1}, s_{j+1}), is tabu at step t if one of the following conditions holds:
  1. $t \leq T_{s_i,s_{i+1}} + L^{\text{out}}$
  2. $t \leq T_{s_j,s_{j+1}} + L^{\text{out}}$
  3. $t \leq T_{s_i,s_j} + L^{\text{in}}$
  4. $t \leq T_{s_{j+1},s_{i+1}} + L^{\text{in}}$

  So, at first all moves are legal
- selected move (i^*, j^*), update the auxiliary structures setting
  1. $T_{s_{i^*},s_{i^*+1}} := t$
  2. $T_{s_{j^*},s_{j^*+1}} := t$
  3. $T_{s_{i^*},s_{j^*}} := t$
  4. $T_{s_{j^*+1},s_{i^*+1}} := t$

As n arcs are in and $n(n-2)$ out of the solution, it is better to set $L^{\text{out}} \ll L^{\text{in}}$

Example: the *Max-SAT*

Consider the neighbourhood $N_{\mathcal{F}_1}$, which includes the solutions obtained complementing the value of a variable (all n solutions are feasible)

Since $|x| = |B \setminus x|$ for each $x \in X$

- the tabu tenure for additions and deletions can be the same
- it is sufficient to forbid the change of value of a variable
  and the attribute is the variable

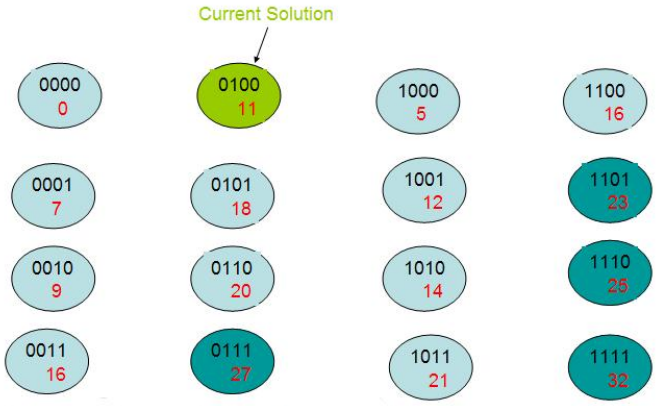
The algorithm proceeds as follows

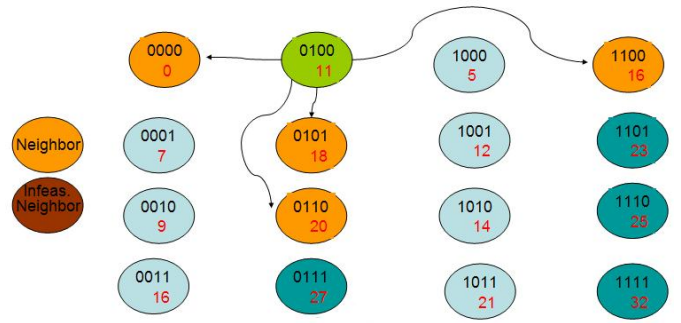
- at first, set $T_i = -\infty$ for each variable $i = 1, \ldots, n$
- at each step t, explore the n solutions obtained complementing each variable
- the move i, which assigns $x_i := \bar{x}_i$, is tabu at step t if $t \leq T_i + L$
  (*at first all moves are nontabu*)
- perform move i^* and set $T_{i^*} := t$

Example: the *KP*

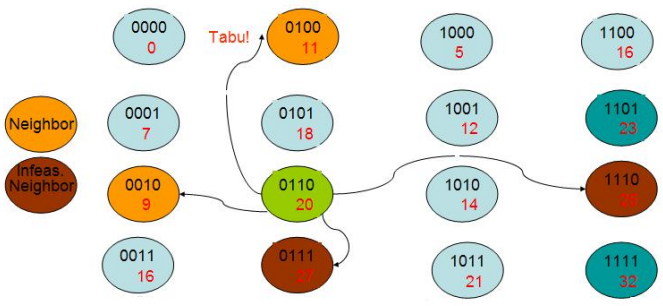
The neighbourhood $N_{\mathcal{H}_1}$ includes al solutions at Hamming distance ≤ 1

For the sake of simplicity use the variable as an attribute (with $L = 3$): vector T saves the iteration of the last move performed on each $i \in B$

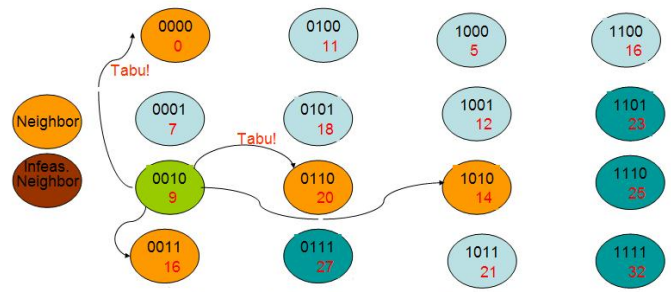




$t = 1 \quad T = [-\infty \ -\infty \ -\infty \ -\infty]$



$t = 2 \quad T = [-\infty \ -\infty \quad 1 \ -\infty]$



$t = 3 \quad T = [-\infty \quad 2 \quad 1 \ -\infty]$

Tuning the *tabu tenure*

The value of the *tabu tenure* L is a crucial parameter

- too large tenures can conceal the global optimum and in the worst case block the search
- too small tenures can hold the exploration back in useless regions and in the worst case produce cyclic behaviours

The most effective value of L is in general

- related to the size of the instance
- slowly growing with size (many authors suggest $L \in O(\sqrt{|A|})$)
- but nearly constant on medium ranges of size

Cycles can be broken extracting L at random in a range $[L_{\min}; L_{\max}]$

Adaptive mechanisms update L based on the results of the search within a given range $[L_{\min}; L_{\max}]$

- decrease L when the current solution x improves: the search is probably approaching a new local optimum and we want to favour it (intensification)
- increase L when the current solution x worsens: the search is probably leaving a known local optimum and we want to speed up (diversification)

Variants

Other adaptive strategies work in the long term:

- reactive Tabu Search:
  - use efficient structures to save the solutions visited (*hash table*)
  - detect cyclic behaviours (frequent repetitions)
  - move the range $[L_{\min}; L_{\max}]$ upwards if the solutions repeat too often
- frequency-based Tabu Search:
  - save the frequency of each attribute in the solution in structures similar to the ones used for the tenure (e.g., F_i for each $i \in B$)
  - if an attribute appears very often
    - favour the moves introducing it modifying f as in the *DLS*
    - forbid the moves introducing it, or discourage them by modifying f
- Exploring Tabu Search: reinitialize the search from solutions of good quality which have been explored, but not used as current solution (*i. e., the "second-best solutions" of some neighbourhood*)
- Granular Tabu Search: enlarge or reduce the neighbourhood progressively